# Multi-object Rearrangement in Confined Spaces using a Car-like Robot Pusher

Jeeho Ahn[1] and Christoforos Mavrogiannis[1]

***Abstract*— We focus on push-based multi-object rearrangement planning using nonholonomically constrained mobile robots. The simultaneous geometric, kinematic, and physics constraints make this problem especially challenging. Prior work often relaxes some of these constraints by assuming dexterous hardware, prehensile manipulation, or sparsely occupied workspaces. Our key insight is that by capturing these constraints into a unified representation, we could empower a constrained robot to tackle difficult problem instances by modifying the environment in its favor. To this end, we introduce a Push-Traversability graph, whose vertices represent poses that the robot can push objects from, and edges represent optimal, kinematically feasible, and stable transitions between them. Based on this graph, we develop ReloPush, a planning framework that takes as input a complex multi-object rearrangement task and breaks it down into a sequence of single-object pushing tasks. We evaluate ReloPush across challenging scenarios, involving the rearrangement of up to nine objects, using a 1/10-scale robot racecar. Compared to two baselines lacking our proposed graph structure, ReloPush exhibits orders of magnitude faster runtimes and significantly more robust execution in the real world, evidenced in lower execution times and fewer losses of object contact.**

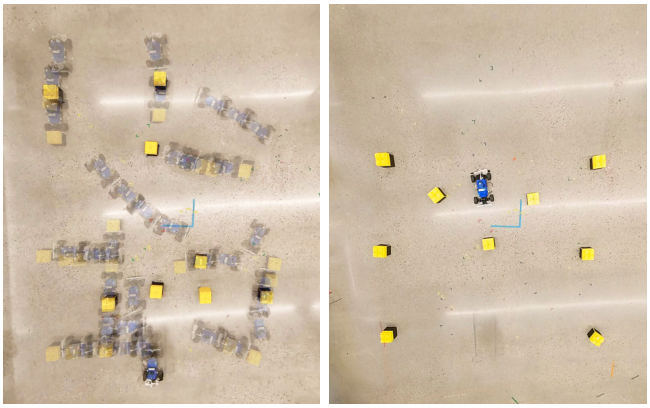
## I. Introduction

Autonomous mobile robots have revolutionized fulfillment by offering a robust and scalable solution for large-scale rearrangement tasks. Fulfillment centers leverage extensive structure: robots often move across rectilinear rail grids, and make use of specialized docking mechanisms. This structure is missing from many other critical domains like construction, waste management, and small-to-medium warehouses. These domains give rise to rearrangement tasks involving objects of various shapes, and navigation among dense clutter while respecting boundary and kinematics constraints.

A practical approach to extending the range of rearrangeable objects is pushing, a class of *nonprehensile manipulation* that handles objects without requiring secure grasping [8]. This technique is appealing, as it enables manipulation of large, heavy, or irregularly shaped objects using relatively simple mechanisms. However, pushing introduces motion constraints: maintaining object stability requires avoiding abrupt turns and excessive accelerations.

To enable navigation among dense clutter in constrained workspaces, prior research addressed such challenges using the paradigm of *planning among movable objects* [13, 14], strategically modifying the environment to facilitate planning. Yet, these approaches typically involve dexterous manipulators capable of unconstrained grasping, neglect orientation constraints on goal object poses, and assume generous workspace boundaries.

[1]Department of Robotics, University of Michigan, Ann Arbor, USA. Email: {jeeho, cmavro}@umich.edu



(a) Executing a rearrangement plan. (b) Resulting rearrangement.

Fig. 1: In this work, we describe ReloPush [2], a planning framework for tackling multi-object rearrangement tasks with a nonholonomic mobile robot pusher.

Here, we focus on multi-object rearrangement via pushing within confined workspaces using a nonholonomic mobile robot pusher. Our key insight is that integrating geometric, kinematic, and physics constraints into a unified representation enables strategic environmental modification, thus facilitating complex rearrangement tasks. To this end, we introduce a *push-traversability* graph, where edges represent kinematically feasible and stable object displacements. Planning on this graph yields effective rearrangement plans for densely cluttered environments (Fig. 1). Extensive hardware experiments, including the creation of room-scale pixel art [1], underscore the robustness of our system. An extended version of this work appears at ICRA 2025 [2].

## II. Problem Statement

We consider a mobile robot *pusher* and a set of $m$ polygonal *blocks* in a workspace $\mathcal{W} \subset SE(2)$. We denote the state of the pusher as $p \in \mathcal{W}$ and the states of the blocks as $o_j \in \mathcal{W}$, $j \in \mathcal{M} = \{1, \dots, m\}$. The pusher follows rear-axle, simple-car kinematics $\dot{p} = f(p, u)$, where $u$ represents a control action (speed and steering angle), and may push objects using a flat bumper attached at its front. The goal of the pusher is to rearrange the blocks from their starting configuration, $O^s = (o_1^s, \dots, o_m^s)$, to a goal configuration, $O^g = (o_1^g, \dots, o_m^g)$. We seek to develop a planning framework to enable the pusher to efficiently rearrange all objects into their goal poses. We assume that the pusher has accurate knowledge of its ego pose at all times, and of the starting configuration of all objects, $O^s$.

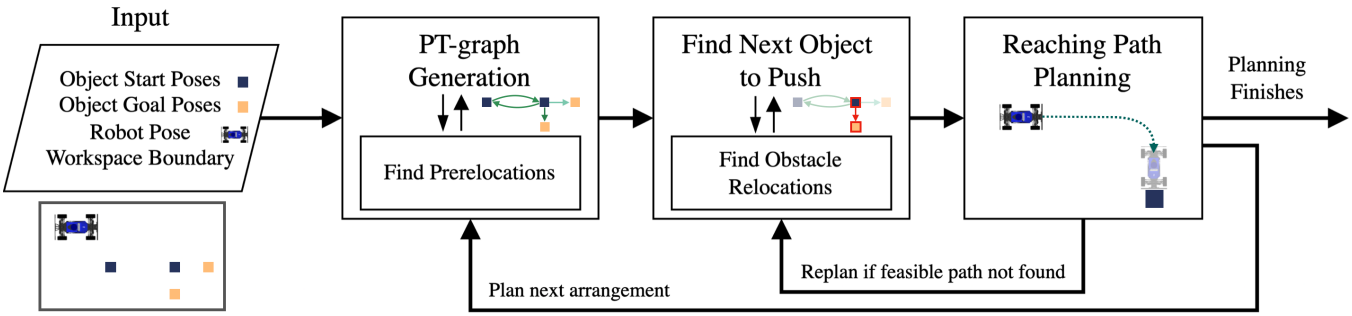
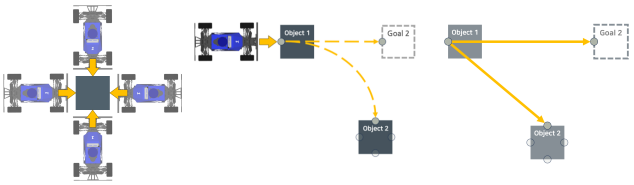
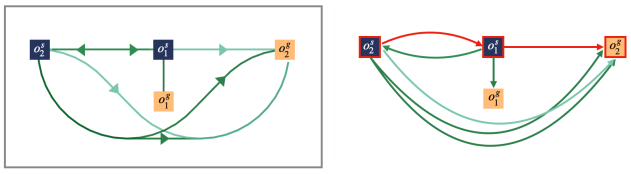



Fig. 2: The ReloPush architecture. Given the initial pose of the pusher and a rearrangement task in the form of start/goal object poses, ReloPush plans an efficient sequence of rearrangement subtasks to be executed by the robot via pushing.




(a) Pushing poses (b) Path Plan (c) PT-graph

Fig. 3: PT-graph generation. (a) First, every object is assigned $K$ pushing poses (e.g., a cubic object has 4 pushing poses). (b) For any pair of pushing poses, we check if a collision-free path that respects the steering limit for quasistatic pushing can be drawn. (c) For each valid path, we construct a directed edge between its start/goal vertices.



(a) Object traversability. (b) PT-graph for the task in (a).

Fig. 4: (a) Two objects (navy squares) need to be rearranged to goal poses (yellow squares). (b) The PT-graph: nodes are pushing poses and edges are Dubins paths connecting them. By searching the graph, we can determine if any blocking objects need to be removed. For instance, the initial pose of object 1 is found to be blocking the shortest rearrangement of object 2 (red path).

## III. ReloPush: Nonprehensile Multi-object Rearrangement

### A. System Overview

Given a workspace $\mathcal{W}$, an initial robot pose $p_s$, and a set of objects that need to be reconfigured from their starting poses $O^s$ to their goal poses $O^g$, ReloPush finds a sequence of rearrangements in a greedy fashion. It first constructs a rearrangement graph (PT-graph) that accounts for robot kinematics, push stability, and workspace boundary constraints. Using graph search, ReloPush searches for the collision-free object rearrangement path of lowest cost. If such a path is found, the graph is updated to mark the rearranged object as an obstacle, and the planner is invoked again to find the next rearrangement of lowest cost. If the path found passes through a blocking object, ReloPush displaces that object out of the way first. If the path violates the workspace boundary, ReloPush displaces the object to be pushed until the path to its goal meets the boundary constraint. If the path is infeasible (i.e., fails to find a motion to approach the object to push), it replans with next rearrangement candidate that has the next lowest cost. This process is repeated until a full rearrangement sequence for all objects is found. An overview of our architecture is shown in Fig. 2.

### B. Push-Traversability Graph

A *traversability* graph (T-graph) is a representation of how movable objects can be reconfigured in a cluttered scene [9]. In its original form, vertices represent (starting and goal) positions of objects and edges represent collision-free transitions between them. By searching the graph, a collision-free rearrangement plan can be found.

Here, we build on the T-graph representation to introduce the *push-traversability* graph (PT-graph) $G(V, E)$, which not only captures the spatial relationships among movable objects but also integrates the kinematic constraints of the pusher and push-stability constraints of objects within the edges. Because in push-based manipulation of polygonal blocks, the block orientation is important, each vertex in our graph $v_i \in V$ represents a valid robot *pushing pose* $p_i$, i.e., a pose from which the pusher can start pushing a block (see Fig. 3).

For each vertex pair $(v_s, v_g)$ representing start and goal *pushing poses*, we construct a directed edge if the optimal Dubins path [3] connecting them is collision-free and within workspace bounds. The optimal path uses left (L), right (R), and straight (S) motion primitives with a minimum turning radius $\rho$ to ensure quasistatic pushing stability [4, 5, 7, 16]. Each valid edge is assigned a weight equal to the path length, with direction dictated by stability constraints (e.g., forward-only pushing).

### C. Prerelocation: Change of Starting Pushing Pose

Often, an edge between two vertices cannot be formed because the connecting Dubins curve violates the workspace boundary, typically due to the limited turning radius $\rho$ required for push stability (see Fig. 5). Our insight is that a slight adjustment of the initial pushing pose can yield an optimal, collision-free rearrangement path within workspace bounds. ReloPush leveraves Dubins path classification [6, 10] to examine the case of the initial Dubins curve for rearrangement, namely *long-path* case or *short-path* case. When the start and goal poses are too closely located that it require large turning, a *short-path* case, ReloPush attempts to find

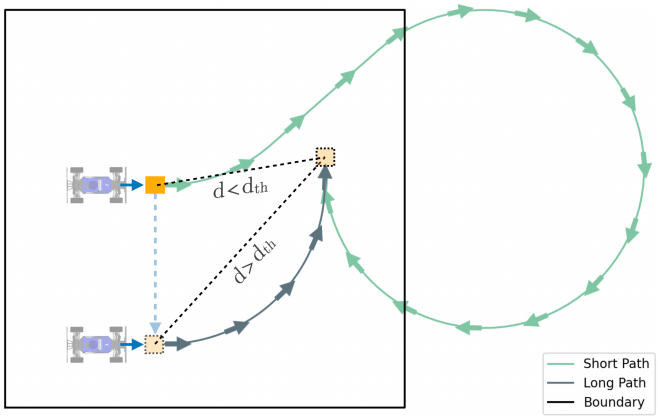


Fig. 5: Two Dubins curves with the same goal pose (top right) and maximum turning radius. When the start pose is too close to the goal ($d \leq d_{th}$), the resulting Dubins curve (green color) is a *Short Path* involving large turns violating the workspace boundary. Using Dubins path classification [6, 10], we can determine a *prerelocation* of the object's starting pose to allow reaching the goal via a *Long Path* ($d > d_{th}$) which will involve smaller turning arcs (gray color).

another pose to start from that makes it a *long-path* case. We refer to this change of starting pose as a *Prerelocation*.

### D. Removing Blocking Objects

Extracting a rearrangement path plan can be done by searching the PT-graph using any graph search algorithm. The extracted path may include a vertex that is different from the start and goal vertex. If that is the case, then that vertex corresponds to an object that is physically blocking the rearrangement path. This object needs to be displaced before the plan can be executed. To do so, we follow a similar technique to how we plan *Prerelocations*: we find the closest relocation along the object's pushing directions (see Fig. 3a) that unblocks the path execution. This method of finding what object to remove is shown to be complete [9].

### E. Analysis of the Algorithm

*Theorem 3.1:* Assuming a bounded number of pushing poses per object, $K_{max}$, the graph construction runs in polynomial time.

*Proof:* The number of vertices per object is bounded by $K_{max}$, thus, a fully connected graph in our case has $K_{max} \cdot m$ vertices. For each edge, a Dubins path is found in $O(1)$ and its collision checking is done in $O(1)$ assuming a bounded number of configurations checked due to our confined workspace. Searching a graph with Dijkstra's algorithm runs in $O(|V|^2)$ in a directed complete graph (the number of edges dominates). Thus, the runtime for a the rearrangement of $m$ objects reduces to $O(m^3)$. To plan motion to approach an object, we invoke Hybrid A*, whose runtime also reduces to a polynomial expression on the number of objects assuming fixed workspace discretization, resolution of driving directions, and replanning attempts. ■

## IV. Evaluation

### A. Implementation

**Experimental Setup**. We implement our framework on MuSHR [12], an open-source 1/10th-scale mobile robot racecar, augmented with a 3D-printed flat bumper for pushing deployed in a workspace of area $4 \times 5.2m^2$. Across simulations and hardware experiments, we assume access to accurate robot localization (in real experiments, we make use of an overhead motion-capture system). We use objects of cubic shape with a side of $0.15m$ and a mass of 0.44 kg. The friction coefficient on the bumper-object surface was measured to be $\sim 0.73$.

**Software**. We implement our framework using the Open Motion Planning Library for Dubins path planning [15], and the code of Wen et al. [17] for Hybrid A* planning. Across simulated and real-world experiments, we use a Receding Horizon Controller (RHC) based on the implementation of the MuSHR [12] ecosystem. We run graph construction and search using Boost Graph Library [11]. We share our software implementation online at **https://github.com/fluentrobotics/ReloPush**.

**Metrics**. We evaluate our system with respect to the following metrics:

- S: Success rate – a trial is successful if a planner successfully finds a feasible rearrangement sequence.
- $T_p$: The time it takes to compute a complete rearrangement plan.
- $L_t$: The total length of the path that the robot travelled, including the reaching and pushing segments.
- $N_{loss}$: The total number of objects the robot lost contact with.
- $T_e$: The total time takes to execute a complete rearrangement plan.

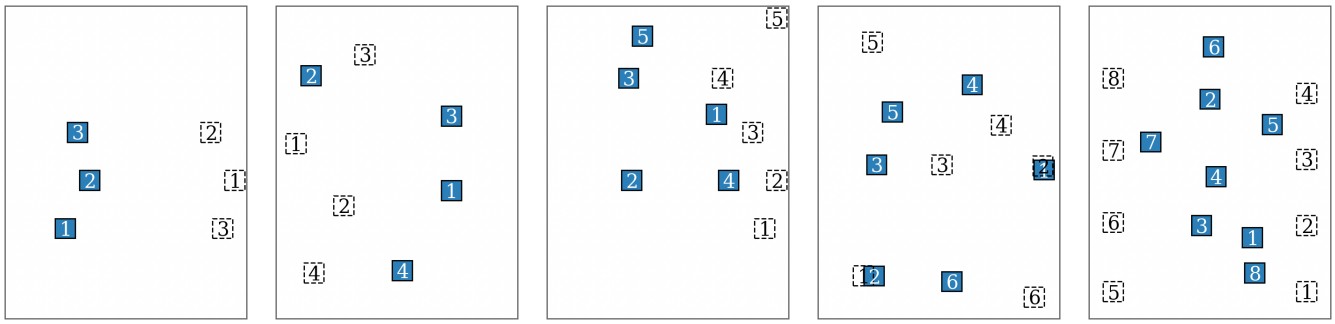
We also extract insights on the planning behavior of all algorithms using the following indices:

- $N_{pre}$: The total number of objects prerelocated to a feasible starting pushing pose (see Fig. 5).
- $N_{obs}$: The total number of removed blocking objects (see Fig. 4).
- $L_p$: The total length of path segments involving pushing.

**Baselines**. We compare the performance of ReloPush against two baselines:

- NO-PRERELO (NPR): a variant of RELOPUSH that also uses the PT-graph to handle nonmonotone cases but does not plan *prerelocations*. Instead, it invokes Hybrid A* if a collision-free Dubins path is found, to add edges.
- MP: a variant of our system that does not make use of the PT-graph at all but rather invokes Hybrid A* to plan a sequence of rearrangement tasks in a greedy fashion, and thus can only handle monotone cases.

**Experimental Procedure.** We consider a series of rearrangement scenarios of varying complexity (see Fig. 6) instantiated in simulation and the real world. To extract statistics on planning performance, we instantiated 100 trials of each scenario by locally randomizing the start and goal positions of objects within a range of $\pm 0.05m$ around the nominal instances of Fig. 6. To evaluate the robustness of our complete architecture, we executed the same scenarios in a physical workspace on a real MuSHR [12] robot. To ensure fairness in real-robot experiments, we chose instances



(a) $m = 3$ (b) $m = 4$ (c) $m = 5$ (d) $m = 6$ (e) $m = 8$

Fig. 6: Evaluation scenarios. Solid squares represent starting object poses and dashed squares represent goal poses.

TABLE I: Planning performance. Each cell lists the mean and the standard deviation over 100 trials per scenario.

| **Scenario** | $m = 3$ | | | $m = 4$ | | | $m = 5$ | | | $m = 6$ | | | $m = 8$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Algorithm** | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP |
| $S$ (%) | **100** | 55 | 52 | **100** | **100** | 73 | **80** | 64 | 56 | **89** | 25 | 3 | **86** | 12 | 6 |
| $T_p$ (ms) | **40** (4.3) | 1864 (141.1) | 465 (67.4) | **86** (3.5) | 5376 (131.5) | 956 (36.0) | **146** (6.8) | 9034 (581.5) | 1175 (117.6) | **318** (20.8) | 14489 (682.3) | 1487 (57.4) | **529** (23.5) | 25654 (2205.6) | 2073 (149.5) |
| $L_t$ (m) | **32.3** (3.1) | 42.1 (5.2) | 38.7 (7.3) | **40.3** (0.8) | 45.5 (2.5) | 45.1 (2.5) | **48.3** (3.5) | 60.8 (6.5) | 61.0 (9.1) | 77.6 (5.5) | 74.5 (15.2) | **62.9 (1.0)** | **90.3** (6.0) | 105.8 (4.2) | 101.6 (3.5) |

TABLE II: Planning behavior. Each cell lists the mean and the standard deviation over 100 simulated trials per scenario.

| **Scenario** | $m = 3$ | | | $m = 4$ | | | $m = 5$ | | | $m = 6$ | | | $m = 8$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP |
| $N_{pre}$ | 0.9 (0.29) | - | - | 1.0 (0.00) | - | - | 1.6 (0.49) | - | - | 3.0 (0.45) | - | - | 4.4 (0.59) | - | - |
| $N_{obs}$ | 0.0 (0.00) | 0.0 (0.00) | - | 0.0 (0.00) | 0.0 (0.00) | - | 0.5 (0.52) | 1.0 (0.25) | - | 1.0 (0.00) | 0.5 (0.50) | - | 0.6 (0.49) | 0.0 (0.00) | - |
| $L_p$ (m) | **8.4** (0.1) | 19.1 (4.4) | 17.8 (4.3) | **8.5** (0.10) | 18.1 (0.8) | 17.3 (0.9) | **11.2** (0.5) | 29.6 (10.1) | 37.6 (7.2) | **13.9** (0.9) | 37.2 (7.8) | 35.6 (0.4) | **23.7** (0.76) | 59.3 (3.1) | 55.9 (4.0) |

TABLE III: Results from real-world trials. Each cell lists the mean and the standard deviation over 5 trials per scenario.

| **Scenario** | $m = 3$ | | | $m = 4$ | | | $m = 5$ | | | $m = 6$ | | | $m = 8$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Algorithm** | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP | RELOPUSH | NPR | MP |
| $N_{loss}$ | **0.0** (0.0) | 0.6 (0.49) | 1.0 (0.0) | **0.0** (0.0) | 0.6 (0.49) | 1.0 (0.63) | **0.0** (0.0) | 0.4 (0.49) | 0.8 (0.75) | **0.2** (0.40) | 1.4 (0.49) | 1.2 (0.40) | **0.2** (0.4) | 2.0 (0.63) | 1.6 (0.49) |
| $T_e$ (s) | **96.6** (0.07) | 114.4 (0.49) | 104.0 (4.23) | **121.7** (0.32) | 137.4 (1.05) | 133.7 (0.37) | **148.1** (0.66) | 154.0 (1.46) | 235.8 (1.71) | 256.2 (3.08) | 198.9 (2.74) | **182.97** (1.20) | **274.4** (0.33) | 314.9 (2.28) | 285.9 (1.40) |

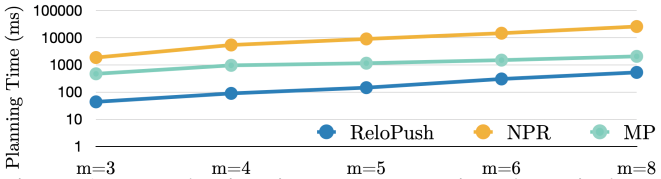
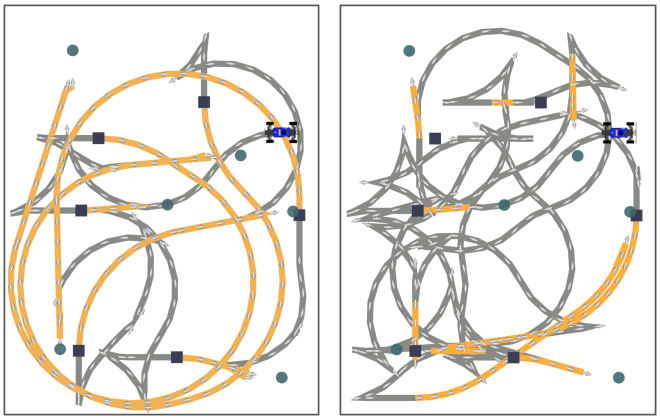



Fig. 7: Average planning time across scenarios, shown in logarithmic scale. ReloPush scales well with the number of objects compared to baselines.

where all algorithms were successful in planning. We ran each scenario 5 times per algorithm.

### B. Results

**Planning Performance.** ReloPush dominates baselines in success rate and planning time (see Table I, Fig. 7). The gap becomes more pronounced as the clutter (number of objects) increases. This happens because increased clutter is more likely to lead to nonmonotone instances. For example, most $m = 6$ instances are nonmonotone because two objects overlap with goals of other objects. Since MP can only handle monotone rearrangements, it fails more frequently in these harder instances. It is also worth observing that kinematic constraints make some instances harder to solve *regardless* of the number of objects. For example, some of the $m = 3$ instances are challenging because $o_2^s$ is situated so close to its goal $o_2^g$ that the optimal path connecting them goes out of the boundary. In contrast, *ReloPush* handled this scenario effectively via *prerelocation*. Table II provides intuition on the planning decisions that ReloPush made enabled increased performance. As clutter increases, ReloPush makes



(a) MP (b) RELOPUSH

Fig. 8: Paths generated by MP (a) and RELOPUSH (b) for the $m = 6$ scenario. Squares and circles represent respectively start and goal object poses. Continuous lines represent planned paths with pushing segments shown in yellow. RELOPUSH plans substantially shorter pushing segments to minimize the risk of losing contact with an object during execution.

increasingly more workspace modifications (prerelocations and blocking-object removals).

**Real Robot Experiments.** ReloPush never lost contact with any objects, in contrast to baselines (see Table III). One reason for that is that ReloPush maintains low pushing path length (for better or similar total path length) as shown in Table II. The shorter the pushing distance, the lower the risk of losing the object due to model errors and uncertainties (see Fig. 8). A video with footage from our experiments can be found at https://youtu.be/_EwHuF8XAjk.

## References


[1] J. Ahn and C. Mavrogiannis. From A to Ω: Pixel art with a mobile robot. In *Special Session: Arts in Robotics, IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[2] J. Ahn and C. Mavrogiannis. ReloPush: Multi-object rearrangement in confined spaces with a nonholonomic mobile robot pusher. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[3] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[4] S. Goyal, A. Ruina, and J. Papadopoulos. Planar sliding with dry friction part 1. limit surface and moment function. *Wear*, 143(2):307–330, 1991.

[5] J. King, M. Klingensmith, C. Dellin, M. Dogar, P. Velagapudi, N. Pollard, and S. Srinivasa. Pregrasp manipulation as trajectory optimization. In *Proceedings of Robotics: Science and Systems (RSS)*, 2013.

[6] J. Lim, F. Achermann, R. Bähnemann, N. Lawrance, and R. Siegwart. Circling back: Dubins set classification revisited. In *Workshop on Energy Efficient Aerial Robotic Systems, International Conference on Robotics and Automation (ICRA)*, 2023.

[7] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *The International Journal of Robotics Research*, 15(6):533–556, 1996.

[8] K. M. Lynch and M. T. Mason. Dynamic nonprehensile manipulation: Controllability, planning, and experiments. *The International Journal of Robotics Research*, 18(1):64–92, 1999.

[9] C. Nam, S. H. Cheong, J. Lee, D. H. Kim, and C. Kim. Fast and resilient manipulation planning for object retrieval in cluttered and confined environments. *IEEE Transactions on Robotics*, 37(5):1539–1552, 2021.

[10] A. M. Shkel and V. Lumelsky. Classification of the dubins set. *Robotics and Autonomous Systems*, 34(4): 179–202, 2001.

[11] J. Siek, L.-Q. Lee, and A. Lumsdaine. Boost graph library. http://www.boost.org/libs/graph/, June 2000.

[12] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, R. Scalise, J. R. Smith, S. Choudhury, C. Mavrogiannis, and F. Sadeghi. MuSHR: A low-cost, open-source robotic racecar for education and research, 2019. *arXiv:1908.08031* [cs.RO].

[13] M. Stilman and J. Kuffner. Planning among movable obstacles with artificial constraints. *The International Journal of Robotics Research*, 27(11-12):1295–1307, 2008.

[14] M. Stilman, J.-U. Schamburek, J. Kuffner, and T. Asfour. Manipulation planning among movable obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3327–3332, 2007.

[15] I. A. Sucan, M. Moll, and L. E. Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[16] S. Talia, A. Thareja, C. Mavrogiannis, M. Schmittle, and S. S. Srinivasa. PuSHR: A multirobot system for nonprehensile rearrangement. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Sytems (IROS)*, pages 5380–5387, 2023.

[17] L. Wen, Y. Liu, and H. Li. Cl-mapf: Multi-agent path finding for car-like robots with kinematic and spatiotemporal constraints. *Robotics and Autonomous Systems*, 150:103997, 2022.